Vehicle Detection, Tracking, and Crash Prediction based on YOLO

Anonymous EECS 442 submission

Paper ID ******

Abstract

In the field of autonomous driving, detecting and predicting crashes may potentially help improve traffic situations and save lives. In this report, we explore a pipeline to track car trajectories at road crossing and predict potential accidents. Our project explores the YOLO model for object detection, algorithms for incorporating angle prediction for YOLO, vehicle identification, trajectory prediction, as well as crash risk assessments. We present an efficient pipeline to conduct crash detection using birds-eye-view input, contributing to the ongoing governmental efforts to improve road safety. Our project has the potential to be implemented in real-world road crossings to provide more agile emergency services in the event of accidents.

Keywords: YOLO, Object Detection and Tracking, Crash Prediction, Transfer Learning

1. Introduction

With modern autonomous driving technologies, we see a significant advancement in vehicular safety and traffic management, primarily through the use of bird's-eye view (BEV) cameras combined with machine learning algorithms. This paper focuses on enhancing the predictive capabilities of these technologies by adapting the YOLO (You Only Look Once) model to better interpret data provided by BEV cameras, thus improving its effectiveness in predicting vehicle trajectories and potential collision points.

Our research investigates modifications to the YOLO architecture to address the challenges of real-time, accurate path prediction and collision detection in dynamic traffic scenarios. Our study aims to contribute to safer autonomous driving systems and provide insights into the integration of advanced object detection technologies in traffic management.

2. Background

Safety measures, particularly in predicting and preventing potential traffic collisions become more and more significant these days. This research is motivated by the urgent need to reduce traffic accidents, which remain a leading cause of mortality globally. Utilizing computer vision to predict vehicular crashes before they occur can significantly contribute to the safety and efficiency of road traffic systems.

Bird's-eye view (BEV) cameras provide a comprehensive perspective of vehicular dynamics, offering a unique vantage point compared to traditional forward-facing vehicle cameras. This allows for more effective monitoring of vehicle trajectory, which are essential in predicting potential collision points. Our approach aims to explore these cameras' potential in the specific task of crash prediction.

The use of datasets like DOTA demonstrates the versatility of machine learning applications. However, adapting these datasets to train models for ground-level vehicular traffic analysis presents unique challenges, including the need for precise object detection and orientation. Our methodology addresses these challenges by refining detection algorithms to suit the specific vehicular situations and crash predictions.

In recent years, models such as YOLO have revolutionized object detection due to their speed and efficiency. However, the application of such models in dynamic, real-time environments like traffic systems requires significant modifications to accommodate the unique aspects of vehicular movements. This research expands on existing knowledge by adapting the YOLO architecture to detect oriented objects and predict their trajectories, thereby anticipating potential collisions more effectively. By enhancing the capabilities of existing computer vision techniques, we aim to contribute to the development of safer autonomous driving technologies and reduce the incidence of traffic accidents.

3. Methodology

To achieve the crash prediction feature we planned, there are a few key steps to conduct: 1. car detection 2. car identi-

fication 3. trajectory prediction 4. determining crashes. Below we discuss implementation details and considerations for each part.

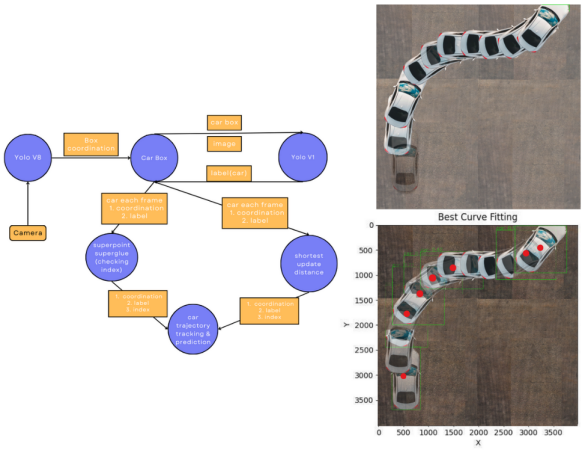

Figure 1. ROS Package Pipeline

Based on the ROS package we created, the user can input the RGB images from the camera, and the ROS package will finally output the car position, the car trajectory and the future driving trajectory prediction as shown in Fig. 1.

3.1. Dataset

Our primary goal is to predict car crashes with a birds-eye view camera live feed. Naturally, the dataset we incorporate in our model training and system design should primarily include birds-eye-view (BEV) images. the DOTA [10] dataset is used for this purpose.

One of the datasets we tried to inspect is the DOTA dataset v1.0, a dataset dedicated to detecting objects from satellite and aerial images. The 1.0 version gives 20 different object categories including genres like basketball court or bridge, but for tracking purposes, only five of them are objects that can move: large vehicle, small vehicle, helicopter, plane, and ship should be taken out and inspected. Thus, one of the good ways to redirect the dataset to our purposes is to only filter out the tags needed. Besides, we will have to remove the images from the dataset that does not contain any labels that we would want.

Another thing we must take into account is the fact that DOTA dataset is annotated under the format "x1, y1, x2, y2, x3, y3, x4, y4, category, difficult", which are the four points of the quadrilateral for the label. For the purpose of OBB detection, we might have to translate the four points into xywhθ formatting during data loading. If we are not interested in orientation in this case, we will have to translate the coordinates into a two-point label system by taking the min and max of x and y coordinates respectively to align with the output from YOLO.

Furthermore, one additional limitation of the dataset is that the size and shape of the input image can vary by a great extent and it would be fundamentally inefficient to shrink the entire image into a 448x448 image as a car might become less than 10 pixels as shown in Fig. 2.

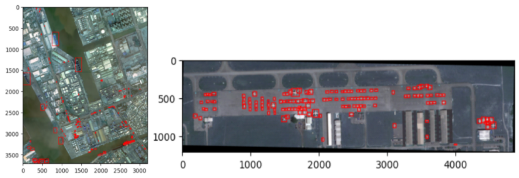

Figure 2. Example Images from the DOTA Dataset

Instead, though it slightly defeats the definition of the YOLO, we can essentially crop the images to smaller samples of it so that smaller objects are easier to be detected given limited input dimensions. Even given the upsampling structures as seen in YOLO v3 [6], the objects shown in the network will still have too few pixels to be detectable even by human eyes. Else, if the width and height of the images are too different, the objects can be distorted as a result.

To fix this, we can crop the images into smaller rectangular sizes like (448, 448) while retaining all the labels, filter out the images we need, and the model will have an easier time recognizing the patterns given the fixed size model. Here are some examples of the cropped samples from the original training set:

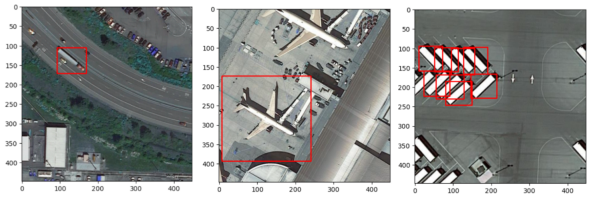

Figure 3. Examples of the Cropped Samples from DOTA

This is an example of how to adapt datasets into different purposes and some considerations before putting everything into training the model. However, during our initial attempts to actually train the network, images from DOTA is still too big for our purposes. Thus, we implemented transfer learning using PASCAL_VOC [2] dataset.

3.2. Car Detection

We chose the YOLO v1 [5] model as the base algorithm for car detection. Due to our more narrow use case, we modified the model output layer to predict fewer classes (5, rather than 20). We also adopted transfer learning techniques, hoping to generalize the model to be resilient in the case of extraordinary input. Finally, due to the oriented nature of the DOTA dataset, we modified the model output

layer definition as well as the loss function to correctly calculate the orientation loss.

3.2.1 Detection Model Architecture

Table 1. Neural Network Architecture

Type	Filters	Size/Stride	Output
Conv	64	7 x 7 / 2	224 x 224
Max Pool		2 x 2 / 2	112 x 112
Conv	192	3 x 3 / 1	112 x 112
Max Pool		2 x 2 / 2	56 x 56
1 x			
Conv	128	1 x 1 / 1	56 x 56
Conv	256	3 x 3 / 1	56 x 56
Conv	256	1 x 1 / 1	56 x 56
Conv	512	3 x 3 / 1	56 x 56
Max Pool		2 x 2 / 2	28 x 28
4 x			
Conv	256	1 x 1 / 1	28 x 28
Conv	512	3 x 3 / 1	28 x 28
Conv	512	1 x 1 / 1	28 x 28
Conv	1024	3 x 3 / 1	28 x 28
Max Pool		2 x 2 / 2	14 x 14
2 x			
Conv	512	1 x 1 / 1	14 x 14
Conv	1024	3 x 3 / 1	14 x 14
Conv	1024	3 x 3 / 1	14 x 14
Conv	1024	3 x 3 / 2	7 x 7
Conv	1024	3 x 3 / 1	7 x 7
Conv	1024	3 x 3 / 1	7 x 7
FC		4096	4096
Dropout 0.5			4096
FC		7 x 7 x 30	7 x 7 x 30
FC		496	496
Dropout 0.5			496
FC		7 x 7 x 15	7 x 7 x 15
LeakReLU		7 x 7 x 15	7 x 7 x 15

The original YOLO v1 architecture we implemented can detect some labels such as person, cat, dog kind of good. However, for the car, mbike, and bus, it has a really bed detection, so in order to solve this problem, we will use the transfer learning method. Tab. 1.

3.2.2 Transfer Learning

We use the YOLO v1 Darknet architecture with 24 layers combined with transfer learning to train our custom computer vision model, specifically for detecting **cars, buses, bikes, mbike, and persons**. Fig. 4The detailed architecture is specified in Tab. 1.

tuned parameters	values
pretrained_dataset	heavy_3000_20.csv
batch size	64
Learning Rate	$f(x) = \begin{cases} 2e^{-5} & \text{if mAP} < 0.65 \\ 2e^{-6} & \text{if mAP} \geq 0 \end{cases}$
transfer learning learnable/freeze	1. change the last 2 fully connection layers to fit in the 5 labels 2. freeze the first 18 layers. 3. make the last 6 CNN and 2 new fully connection layers be learnable.
transfer learning dataset	transfer_1000_5.csv

Table 2. Transfer Learning Parameters

We initially utilized the original YOLO v1 architecture along with 3000 randomly selected images from 20 labels to train the model on drawing bounding boxes and identifying large objects within each frame. However, given our project's emphasis on predicting car trajectories and issuing collision warnings, we use transfer learning to let the model specifically learn on these five labels. The parameters tunning and transfer learning detains are in Tab. 2

To accommodate the prediction of five classes, we modify the output layers from 7x7x30 to 7x7x15 as $15 = 2 * 5 \cdot |\text{bbox\_params}| + 5 \cdot |\text{labels}|$.

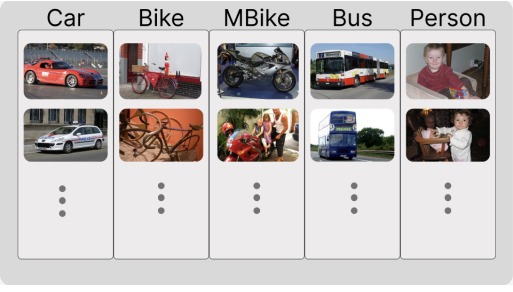

Figure 4. Five Classes

3.2.3 Complex (Oriented) YOLO Loss

To detect car bounding boxes that are actually oriented, we incorporate oriented bounding box notation, where a bounding box is described not by (x, y, w, h) in YOLO [5], but (x, y, w, h, θ). Essentially, besides the center, width, height of a bounding box, an additional parameter θ is added to describe the heading of the bounding box. This modification to the YOLO loss is presented in Complex YOLO [9]. Complex-YOLO proposed L_{Euler}, the loss related to the bounding box angle, described with complex numbers. Formally, this part of the loss is defined as

$$\begin{aligned}\mathcal{L}_{Euler} &= \lambda \sum_{S^2} \sum_{B} \mathbb{1}^{obj} |e_{pred} - e_{gt}|^2 \\ &= \lambda \sum_{S^2} \sum_{B} \mathbb{1}^{obj} [(r_{pred} - r_{gt})^2 + (i_{pred} - i_{gt})^2]\end{aligned}$$

where e_{pred}, e_{gt} are complex numbers on the unit circle representing the angle of predicted and ground-truth bounding box, respectively, and r, i are their real and imaginary parts, respectively. Thus, the Complex YOLO loss is defined as

$$\mathcal{L}_{Comp} = \mathcal{L}_{Yolo} + \mathcal{L}_{Euler}$$

We ultimately ran out of time to modify our model output and train on this modified loss, but it is certainly worth exploring as an extension to the traditional YOLO algorithm.

3.2.4 Oriented Bounding Box Detection and IOU

One part of the YOLO loss requires the computation of the Intersection-Over-Union (IOU) between the predicted bounding box and the ground truth. The IOU between two bounding boxes is defined as

$$IOU = \frac{Area(B_{pred} \cap B_{gt})}{Area(B_{pred} \cup B_{gt})}$$

While it is straightforward for axis-aligned boxes, computing the IOU of two oriented bounding boxes are much more complex. We developed **PolygonIntersect()** to accomplish this objective. This algorithm computes the polygon intersection of two arbitrary convex quadrilateral in $O(n^2)$ time.

While traditional computer graphics often use the ray casting algorithm [8] to check whether a point is within a polygon, we observed that for a point to be inside a convex polygon, the vector it forms with the vertices of the polygon must have angles of the same sign with respect to corresponding edges. This information is easily attainable through a 2D cross product (or determinant). This is the theoretical foundation of Algorithm 2. While the complexity of Algorithm 2 remains $O(n)$, same as the ray-casting algorithm, this could be more computationally efficient when implemented in vectorized form in a computer system.

Algorithm 1 Polygon Intersection

function INTERSECT($G_1 = (V_1, E_1), G_2 = (V_2, E_2)$)
  Initialize $I \leftarrow [], c \leftarrow Arr[|E_1|, |E_2|]$
  for $(e_1, e_2) \in (E_1, E_2)$ **do**
    if e_1, e_2 not parallel or overlapping **then**
      $c_{ij} \leftarrow$ Intersection(e_1, e_2)
    end if
  end for
  $I \leftarrow \{c_{ij} : c$ lies on $e_i, e_j\}$
  $P_{1\text{in}2} = \{p_1 \in V_1 :$ Within$(G_2, p_1)\}$
  $P_{2\text{in}1} = \{p_2 \in V_2 :$ Within$(G_1, p_2)\}$
  $I \leftarrow I \cup P_{1\text{in}2} \cup P_{2\text{in}1}$
return I
end function

Algorithm 2 Within

function WITHIN($G = (V, E), p$)
  $S \leftarrow Arr[|E|]$
  for $(e \in E$ **do**
    $x_1, y_1 \leftarrow$ begin(e), $x_2, y_2 \leftarrow$ end(e)
    $x_p, y_p \leftarrow p$
    $vec_e \leftarrow (x_2 - x_1, y_2 - y_1)$
    $vec_p \leftarrow (x_p - x_1, y_p - y_1)$
    $S_i \leftarrow sign(vec_p \times vec_e)$
  end for
return $(S_i >= 0, \forall i \in |E|) \vee (S_i <= 0, \forall i \in |E|)$
end function

Upon finding the description of the polygon, we then proceed to finding the area of the polygon intersection. We implement the Area of Polygon algorithm [4] to compute the area of an arbitrary (non-self intersecting) polygon in $O(|V|)$ time.

This suite of algorithms allows us to compute the areas, thus the IOU, of arbitrary convex quadrilateral intersects, which is perfect for the task and dataset we chose. Results and visualization of this algorithm is presented in the Results section.

Unfortunately, we ultimately were not able to successfully modify our model and corresponding loss function to train the model properly. Thus, we only presented results from the transfer learning model using the PASCAL_VOC dataset and the original YOLO loss.

3.2.5 Detection Classification

Due to the limitations of YOLO v1 that we discovered towards the end of the project, we were unable to use YOLO v1 to detect small objects from a large BEV image. To accommodate this, we tried to incorporate the YOLO v8 [3] model into the detection pipeline. While the pretrained YOLO v8 model fails to classify top-down views of cars

most of the time, our transfer-learning model is able to successfully detect this if the image is cropped out.

Thus, for our final version of the prototype, we first detect objects with the YOLO v8 model, which most of the time correctly bounds the box (but produces a wrong label, like phone), crops this part of the image out, and then feed this into our YOLO v1 transferred model to determine whether the cropped image is a car or not. This method bears much improvement but it demonstrates that our transfer-learning demonstrates an improvement over even the vanilla YOLO v8 on specific tasks like top-down-view vehicle detection.

3.3. Trajectory Prediction

To predict the trajectory of all cars detected, we must conduct two tasks: 1. Identify each car and make sure the waypoint is correctly assigned to the same car if previously detected. 2. Predict the trajectory given previous waypoints detected.

3.3.1 Car Identification

We combined k-Nearest Neighbor with the SuperPoint [1]-SuperGlue [7] feature-fitting algorithm to select the car that is not only close to our predicted point, but also resembles the original car the most in the new image, as seen in Algorithm 3.

Algorithm 3 CarID

function CARID($C \in \mathbb{R}^{n\times 2}, P \in \mathbb{R}^{n\times 2}, I$)
  $ID \leftarrow Arr[n]$
  for $c_i \in C$ **do**
    Candidates $= kNN(C_i, P)$
    $ID[i] \leftarrow \text{argmax}_i SPSG(\text{Candidates[i], I})$
  end for
return ID
end function

3.3.2 Trajectory Prediction

We attempted to fit polynomial lines to known waypoints of a car in order to predict its trajectory in the near-future. To achieve this, we first find the best fit polynomial with order no more than 3 using the mean squared loss:

$$\text{argmin}_i \frac{1}{n}\sum_{i=1}^{n}\frac{(y_i - \text{BestFit}(P,i)(x_i))^2}{2}$$

Then, the parametrized best fit polynnomial is used to predict the next waypoint based on the car's current velocity. We use NumPy to compute our best-fit polynomials on the past 4 waypoints (fewer if car just entered frame).

3.4. Crash Detection

To compute whether two cars may crash, we uses an algorithm structure as shown in Algorithm 4 to detect whether the distance between the cars, based on the predicted path, will be lower than a danger threshold value. Way-points are sampled from the provided polynomials at fixed time intervals for the near future, and distance values are computed to determine the risk of crashing.

Algorithm 4 Crash Detection

function CRASHDETECT($Poly_1, Poly_2$)
  $P_1, P_2 \leftarrow \text{Sample}(\text{Poly}_1), \text{Sample}(\text{Poly}_2)$
  $D = \min_t |P_{1,t} = P_{2,t}|^2$
return $D \leq$ DangerThreshold
end function

4. Results

4.1. Car Detection Model

YOLO v1 cannot detect small objects in the frame, at the same time, the YOLO v8 model cannot recognize the top-down view of the car as a car. So we finally use YOLO v8 to generate the box, and then based on the position of the bounding box, extract the cars from the image and pass them into the YOLO v1 model for recognition, ultimately determining their labels.Fig. 5

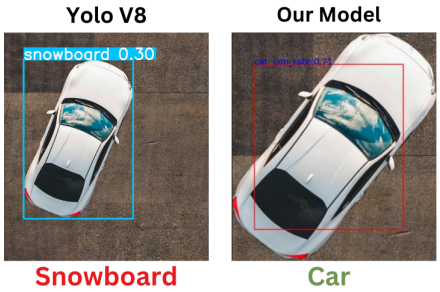

Figure 5. YOLO V8 VS Our Result

4.1.1 Model Training and Transfer Learning

After transfer learning, our model can effectively recognize the five desired labels, while ignoring any objects outside of these five categories. The comparison of performance before and after transfer learning can be visually observed in Fig. 6.

In Fig. 6, the model with pre-trained weights as shown on the lower portion of the figure produced observably more accurate results in recall, classification accuracy, and bounding box accuracy. Given limited hardware capacities, transfer learning will be a much easier approach to

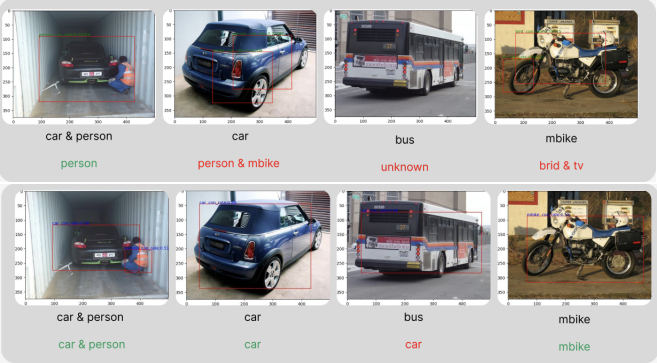

Figure 6. Transfer Learning Results Comparison

4.1.2 Oriented Bounding Box Detection

For Oriented Bounding Box (OBB), our oriented_iou implementation successfully solves for intersects between bounding boxes, their respective areas, as well as the IOU values. It also successfully identifies cases of inclusive, disjoint, and axis-aligned boxes, as shown in Fig. 7.

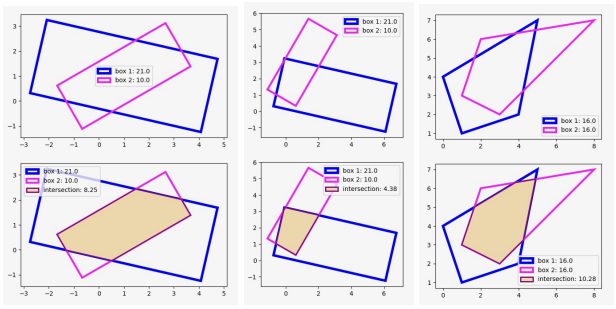

Figure 7. Oriented Bounding Box Detection and Calculation (left: two oriented quadrilaterals, right: detected intersection polygon)

4.2. Car Identity

As shown in Fig. 8, by comparing two photos taken at different times of the same intersection, we employ the SPSG algorithm to perform feature point matching, using white cars and gray cars as examples. We can observe that the white car in the bottom right corner of the second photo belongs to the same vehicle as the white car in the bottom right corner of the first photo, rather than being the same as the gray car in the top left corner of the first photo

However, our model may also have vulnerabilities in cases where two cars of the same make and color, traveling at nearly identical speeds, intersecting trajectories, and both the KNN and SPSG algorithms may fail simultaneously, leading to errors in car index recognition. However, such instances did not occur in our subsequent test cases. The probability of such occurrences is extremely low.

Figure 8. SPSG matching result

4.3. Trajectory Prediction

Our algorithm successfully demonstrated resilience against the extraordinary scenarios discussed in the methods section. In most cases, KNN can correctly identify the car index, but in some extreme situations, it may make the wrong detection and prediction. For example, in Fig. 9, when it comes to 4th frame, relying solely on the KNN algorithm would incorrectly identify the position of the car in the top left corner as that of the car in the bottom right corner. After introducing the SPSG algorithm, discretion between rooftop patterns of two cars result in correct identification, as seen in Fig. 10.

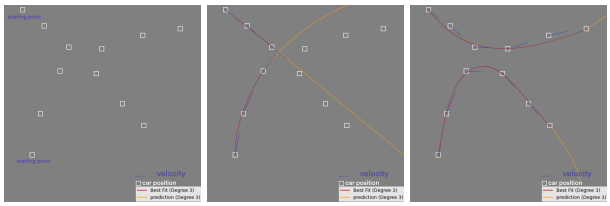

Figure 9. Wrong Prediction

Fig. 11 is an example of six frames where two cars are simultaneously traveling in different directions. Through our detection, recognition, and trajectory prediction pipeline, we can successfully detect the positions of vehicles and predict their trajectories in the near future.

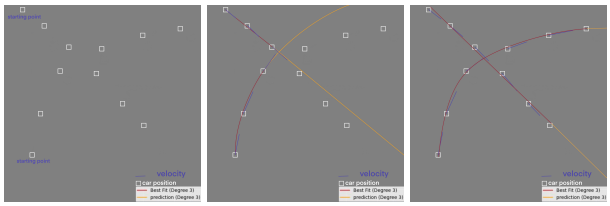

Figure 10. Correct Prediction

5. Conclusion

This paper presents attempts to explore in the task of autonomous driving, focusing on vehicle detection, tracking, and crash prediction using an adapted version of YOLO

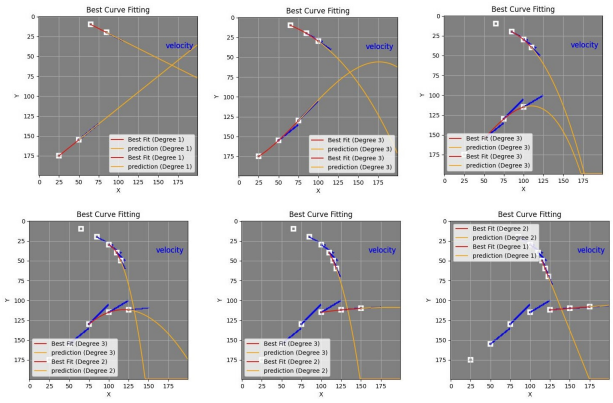

Figure 11. Trajectory Prediction Plots

model. Our modifications to the YOLO architecture, particularly the inclusion of oriented bounding boxes and the integration of a complex YOLO loss, have demonstrated the potential to enhance the accuracy of vehicle trajectory prediction and crash detection.

Moreover, transfer learning has refined our model's performance without relying on extensive datasets, allowing it to more accurately identify and track vehicles under road conditions. The implementation of algorithms for oriented bounding box detection and the calculation of intersection-over-union for complex polygons have provided robust tools for enhancing detection accuracy. Future work will focus on further optimizing the detection speed and reliability, as well as expanding the model's capabilities to include more diverse traffic scenarios and environmental conditions. Please refer to https://github.com/yuzhench/EECS442_final_project.git for more information.

References

[1] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superpoint: Self-supervised interest point detection and description. *CoRR*, abs/1712.07629, 2017. 5

[2] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010. 2

[3] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics YOLO, Jan. 2023. 4

[4] John Page. Algorithm to find the area of a polygon. https://www.mathopenref.com/coordpolygonarea2.html. Accessed: 2024-4-29. 4

[5] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015. 2, 4

[6] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018. 2

[7] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superglue: Learning feature matching with graph neural networks. *CoRR*, abs/1911.11763, 2019. 5

[8] M. Shimrat. Algorithm 112: Position of point relative to polygon. *Commun. ACM*, 5(8):434, aug 1962. 4

[9] Martin Simon, Stefan Milz, Karl Amende, and Horst-Michael Gross. Complex-yolo: Real-time 3d object detection on point clouds. *CoRR*, abs/1803.06199, 2018. 4

[10] Gui-Song Xia, Xiang Bai, Jian Ding, Zhen Zhu, Serge J. Belongie, Jiebo Luo, Mihai Datcu, Marcello Pelillo, and Liangpei Zhang. DOTA: A large-scale dataset for object detection in aerial images. *CoRR*, abs/1711.10398, 2017. 2